A next generation Decentralized Financial System

DEFIS White paper
JUNE 10, 2020

DeFiS is a Decentralized Financial System of scalable, privacy oriented off-chain smart contracts and financial protocols build on top of cryptographic primitives. Owners of different types of cryptocurrencies and crypto-assets can earn interest in DeFiS system, borrow, exchange and create custom cryptocurrencies in a decentralized mode. The technologies of DeFiS improves upon current existing DeFi models and eliminates the limitations and scalability of issues that face the current existing dApps. DeFiS has its own cryptocurrency $XGM, which is the fuel that powers the DeFiS Blockchain

Contents

Challenges of Traditional Financial Market

Financial services, as a rule, make up 20-30% of the total income of the services market and about 20% of the total GDP (Gross Domestic Product) in developed countries. This huge industry basically has a simple idea - to provide a trusted intermediary in transactions between the parties and get your percentage for it. Such a system is not limited to just one intermediary but imposes a whole chain of intermediaries. You can highlight the fundamental problems in the market:

- High transaction costs in modern financial services
- Slow operations, particularly for international transactions
- Excessive bureaucracy
- Lack of transparency for small investors and clear benefits for major players
- Inaccessibility or higher cost of services for a small investor (overhead costs and fees as a percentage are much higher than with large investments)

Decentralized DeFi Solution

Fintech companies are attempting to solve these problems and have achieved some success in this. Fintech solutions such as Internet services, online investment programs, as well as mobile payments have significantly improved the situation. Nevertheless, they are built on top of a system which basically has large intermediation costs. Even if it turns out to reduce some of them, then it will not be possible to solve the problem of lack of transparency since they are built on the same principles and use the same institutions as traditional finance.

The real solution to the problems is the technology of the distributed blockchain registry. Decentralization of financial services can significantly reduce transaction costs in conducting cash transactions and eliminates the need for third parties in the market. Smart contracts for blockchain today can replace most of the functionality of this industry, significantly increasing the return on investment and reducing the costs of market participants. For this reason, many investors began to invest in the cryptocurrency market.

Most crypto investors today use only one way to profit from their capital: raising asset prices. Although in the short term this may be a good investment, it is not how this system can actually work. Investors should be able to lend, invest and profit from their investments, while investing should ensure a return on investment.

Therefore, well-developed decentralized finance systems should allow the creation of various safe and reliable financial instruments for investing in the cryptocurrency itself. Cryptocurrency, by definition, has full transparency compared to traditional systems. Most of the tasks performed by third parties in the chain of traditional financial transactions should be written in the code of decentralized financial systems. Of course, there is some overhead for creating and maintaining the code, as well as network maintenance, but the amount of bureaucracy is minimal, and most of the cost of transactions on such systems is eliminated.

DeFi Market Challenges

Today, the decentralized finance market is the fastest growing and trending sector of the market in the crypto industry. However, further development without solving the following fundamental problems in technology is impossible:

- Lack of scaling smart contracts, which are the basis of any DeFi application
- Lack of confidentiality of the results of the implementation of smart contracts and financial transactions in DeFi applications
- Support for Turing-complete smart contracts. All network resources do not focus on those areas that are necessary in the work of DeFi services
- Actual centralization of applications that do not have their own blockchain, total dependence on the parent blockchain and decisions of third parties
- Consensus issues of existing networks encountered in the development of DeFi applications:
  - Low scalability of networks with POW (Proof-of-Work) consensus
  - Vulnerability of networks with consensus POS (Proof-of-Stake). The inevitable competition of DeFi services with the POS protocol itself in such networks (interest rate competition between DeFi services and POS staking) will lead to vulnerabilities, attacks and centralization of networks

The implementation of a single blockchain system of DeFi services, with technologies that will solve all these problems, will attract huge attention of users around the world and will have significant superiority over analogues.

DeFiS Solution

The project proposes the creation of a single DeFiS system for the provision of decentralized financial services. The system is based on its own specialized blockchain with the ability to scale smart contracts and increased privacy. The technology provides the full necessary functionality for the work and further development of the decentralized finance market, solving the fundamental problems of today's DeFi systems.

Scaling Smart Contracts

Description of the problem: Any DeFi application is based on a smart contract. Bitcoin, Ethereum and other blockchain networks have scripting languages (examples of Script, Solidity, Move, etc.) which are a way of describing the conditions under which "coins" can be spent, which allows you to create smart contracts. To execute such smart contracts, it is necessary that the entire network receives and runs encoded logic. Thus, contracts cannot really be compressed or aggregated in any consistent or consistent manner on existing DeFi networks.

The Solution: DeFiS was created on a fundamentally different Mimblewimble blockchain protocol, and does not use scripts, but operates on Schnorr signatures and Pedersen commitments, which are unspent transaction output amounts spent by UTXO. Our solution for scaling smart contracts is based on Scriptless Scripts and that permanent cryptographic signatures can indirectly communicate something that is not part of the transaction containing the signature. In other words, when someone signs a confirmation of a normal transaction, it is understood that a contract outside the blockchain will correctly execute it. To ensure that such contracts use only signatures, we take advantage of the aggregation of Schnorr signatures. Aggregation is a property that we get from Schnorr's signatures, since they are linear. This means that they can be added and subtracted, and the result is a valid signature corresponding to the same addition or subtraction of public keys. To execute smart contracts without scripts, the conditions for spending coins are determined not by the blockchain, but by the parties themselves. Only after the parties within the framework of the agreement agree on the fulfillment of the conditions, they will begin to cooperate and sign the final transaction.

Confidentiality

Description of the problem: Data not only of transactions but also of the results of smart contracts can be publicly copied and processed. This data is available to all network participants. Having received such data, you can find out how much a certain account received a loan and at what interest rate. Also, this architecture is bad for the interoperability of assets. If for some reason a smart contract is not popular, the investment funds participating in it, publicly displayed on the blockchain, lose their reputation.

The Solution: DeFiS relies on its blockchain built using the most modern technologies and provides transaction confidentiality at the level of cryptographic primitives. The DeFiS blockchain of the project does not store transactions and addresses, but relies on the Confidential Transaction (CT) technology and uses blinding factors to encrypt all the Inputs and Outputs together with public and private keys. Only two sides are dedicated to transaction details. The sec256k1 elliptic curve, Pedersen's commitments and bulletproof, Schnorr's signature and blinding factor, Dandelion ++ and CoinJoin - all these methods, cryptographic primitives and signatures work in a single bundle to ensure the highest level of network privacy. And thanks to the technology of smart contracts built on "Scriptless scripts", the results of smart contracts execution look like a usual signature on the blockchain, and only participants know the details.

Code Security

Description of the problem: Today, most DeFi applications are built on Ethereum, Tron, EOS and many other blockchains that provide Turing-complete smart contracts, most of the functions of which are not required for the development of DeFi applications. At the same time, the necessary functions such as multisig addresses are either not explicitly available or have a complex structure. As smart contracts become more complex, they begin to pose a security risk. For example, ancillary applications may interpret the details of contracts in a slightly different way, which makes it harder to maintain consensus between all network nodes. Potential errors in smart contracts are also public, and this increases the risk of hacker attacks. Such errors led to the hacking of the DAO investment project as well as to the blocking of Parity Technologies funds and other attacks.

The Solution: DeFiS contains the necessary and sufficient set of smart contract functions for implementing DeFi applications of any complexity and is not Turing-complete at the same time. The limitation of the functions of writing smart contracts provides protection against attacks by eliminating possible vulnerabilities in the code.

Decentralization

Description of the problem: Almost all DeFi applications do not have their own blockchain and are "locked" within the parent network on which they are created, forced to unconditionally make any decisions and updates from the developers of the parent blockchain. DeFi applications are completely dependent on third parties interested primarily in their own network, not in their development.

The Solution: DeFiS is launched on its own blockchain, specialized for working with financial services and applications that require scalability and privacy at the level of primitives. All decisions and updates are made by the system participants in the interests of the project.

Consensus issues

- The Vulnerability of networks with Proof-Of-Stake consensus
- The Low scalability of Proof-Of-Work networks

Explanation: POS networks works only when participants are interested in freezing their assets to operate the network, making a profit. But if they can get higher profits by transferring and blocking their

share of assets in the DeFi application, then it is reasonable for them to do this. Then, DeFi services will literally compete with the POS protocol itself. This will lead to the centralization of such networks, attacks and protocol changes for the personal goals of the attackers. This is not valid for DeFi services. All blockchain networks for DeFi applications today are POS networks, with the exception of POW Ethereum (which is the leader today in the DeFi services sector), which will switch to POS this year to solve the scalability problem of POW networks.

The Solution: DeFiS is built on a public blockchain with a POW consensus within which we solved the scalability problem. For increase network scalability, we developed and publicly launched the unique XGM blockchain network with the parallel chain's architecture on which DeFiS runs. The technology allows you to create Confidential Assetchain - parallel, independent blockchain networks that can be combined into a single network of several blockchains. This scalability solution links Confidential Assetchain's together with the Atomic swap protocol. At a basic level, assets can be freely moved from one chain to another using a lock / redeem system. These networks can work independently and not be tied to the main blockchain, which solves the scalability problems for DeFi applications in the project.

DeFi Market

DeFi (Decentralized Finance) has become one of the most significant development areas for the Ethereum platform - in 2019, more than 100 projects created applications and protocols.

According to DeFi Pulse, an analytical site that tracks decentralized finance statistics in general, the DeFi ecosystem in 2019 reached the total number of blocked crypto assets in decentralized financial instruments totaling $ 600 million, but already in early 2020 the figure doubled, reaching a peak in $ 1.2 billion. Analysts at one of Blockchain Capital's largest venture capital funds predict asset growth in the decentralized finance ecosystem in 2020 to $ 5 billion.

DeFiS main features

Decentralized landing and loans

Decentralized lending allows individuals and groups to borrow and lend without the intervention of a bank. Using collateral systems, decentralized lending on the Ethereum platform in February 2020 reached volumes of almost a billion USD. These Ethereum-based systems address only 10% of the market of the total market capitalization of crypto assets. The DeFiS system will provide work, accessing 85% of the market from the start, allowing you to work with most of the top cryptocurrencies in one system.

The main existing decentralized lending platforms (Maker, Compound, Aave, dYdX, etc.) today provide loans at rates up to 12%. Since the logic is controlled through smart contracts, the overhead of banks is eliminated, and platforms are able to offer much better rates than banks.

Collateral lending

The initial DeFiS decentralized lending solution will be fully secured by collateral, and due to the volatility of the cryptocurrency, a double collateral for loans is required. This allows users to take loans in cryptocurrency on the security of another that they own. Thus, they can direct cash flows to solve their problems without the need to sell their cryptocurrency portfolio, and at the same time they can receive favorable loan conditions.

Uncollateralized loan

In the future, it will be possible to provide unsecured loans based on the reputation and other factors about borrowers. Using various forms of verifiable powers, as well as records on the history of borrowing and repaying a loan, systems can be developed without collateral. Many of today's identity solutions that are being developed today focus on anonymous and pseudonymous reputation reporting systems based on a decentralized identifier (DID) issued to an individual and verifiable credentials (VC) issued by well-known authorities that are authoritative to provide information about credit history of an individual.

Appropriate reputation-based systems and risk assessment systems will need to be developed. Such systems will be able to complement or replace today's credit rating ratings.

DeFiS Decentralized Exchange

Decentralized exchanges allow atomic exchanges between assets in various blockchains in p2p mode, allowing users to trade directly, without having to buy and sell cryptocurrency through exchanges. Using a decentralized exchange reduces the risks associated with the use of exchanges and ensures that crypto assets are fully managed by the owners.

Unlike existing decentralized exchanges, the DeFiS solution will integrate atomic swap capabilities into third-party applications by creating a **decentralized exchange** as a **SaaS** service.

Coinization of Assets

Asset tokenization (in our case coinization, mean creating independent blockchain for assets) is the representation of an asset, such as real estate or company capital, in immutable crypto assets on the blockchain. This particular area of decentralized financing has enormous potential and is one of the most interesting investment areas for cryptocurrency holders. A unique feature of DeFiS is that the new crypto assets are not tokens locked in the parent blockchain and depend on it, but independent cryptocurrencies on their own confidential asset chain with unlimited scalability.

DeFiS will provide a module specifically designed for asset co-ordination, and will be especially easy to use for working with real-life assets such as company equity, real estate and other securities.

Transferable Debts and Receivables in DEFI's

DEFI's will offer a set of tools for dealing with transferred debts and receivables. In a centralized financial world, debt and receivable management is only possible through financial institutions. The lack of transparency of these transferred debts was one of the factors that led to the 2008 financial crisis.

For small and medium-sized enterprises, this can be a particularly powerful tool.

Blockchain adds transparency to the exchange of debts and loans based on receivables or other types of financial promises. DEFIS will include the ability for organizations to create smart contracts that allow direct investment in such assets, so that p2p loans can be made without the need for financial intermediaries.

Stable coins

The ability to create decentralized stable coins without collateral. The success of DAI and MakerDAO shows the desirability of binding stable coins, but the high level of support is a deterrent to creating more such

projects. At DeFiS, decentralized, uncollateralized stable coins can be created using market mechanisms and asset blocking.

Dividend distribution / co-investment

Any co-invested asset with a return on investment will be able to use the DeFiS dividend distribution module to create smart contracts that automatically pay investment income. Using this technology will make a leap in the functionality of the distribution of dividends. It will be possible to introduce models similar to traditional finance, in which payments are made weekly, monthly or quarterly, or even every minute.

For example, a government may issue bonds to invest in a wind turbine to provide electricity. The government will take care of everything and pay off this bond in accordance with the schedule. With the DeFiS dividend distribution system, residents could directly purchase a wind turbine and distribute dividends among investors. Instead of going through the necessary administrative procedures through a central authority (government), each user who would like to invest in this wind turbine would do this and receive dividends in accordance with his contribution.

Elimination of overhead costs and fair distribution of profits would be of great benefit to society. In this example, a wind turbine is a public good, but it could also be just a private investment.

The need for co-investment is becoming increasingly relevant with the Internet of things. Devices are capable of creating tremendous value. For example, a car with an autopilot will be able to provide taxi services. Vending machines, sensors, satellites, etc. - all these are potentially profitable devices that people can own together and jointly manage profits, but so far, the legal and financial complexity of this activity has not made it possible to develop such systems. DEFIS can simplify and automate these processes.

DeFiS Technologies

Mimblewimble protocol

DeFiS [XGM] uses Bitcoin's Unspent Transaction Output Model (UTXO). According to this model, there are three types of information that must be hidden in order to make a transaction confidential — sender information, recipient information, and transaction amount. Mimblewimble protocol itself achieves this goal using the following two encryption methods:

- Confidential Transactions
- CoinJoin

But Mimblewimble alone does not counteract all types of blockchain analysis. If implemented naively, Mimblewimble leaves room for peer-to-peer network type analysis, which is very similar to traditional blockchain analysis. That's why in addition to these MW methods, aggregated transactions as they spread across the network, are combined using Dandelion++ of peer-to-peer obfuscation technique in XGM. During the Dandelion stem phase, before the TX's are broadcast to the P2P layer, they are being combined together (CoinJoin). And an addition to this, Decoy Outputs are used, which are added if necessary. XGM is fundamentally private. And it's great! But you can ask — what about transparency? How I can prove a transaction to a third party? XGM is optionally transparent. Payment Proof feature — a cryptographically secure way to prove that a certain transaction really occurred. You can copy payment proof code from your wallet and give it to someone to check (in another wallet) this transaction details (sender, receiver, amount).

Cryptographic primitives

Secp256k1 Library

Cryptographic primitives in DEFIS blockchain are based on the optimized fork of C library for EC operations on elliptic curve secp256k1 that used in bitcoin (3rdparty/secp256k1-zkp).

The secp256k1 naming mean:

"SEC" to denote "Standards for Efficient Cryptography" (used SEC2 2.7.1)

"P" denoting the use of parameters over a prime field Fp, the p is followed by a number

"256" denoting the length in bits of the field size p, that suggests the difficulty of solving the DL on the curve

"K" to denote parameters associated with a Koblitz curve9, to be distinguished from an r, that would denote the use of verifiable random parameters

"1" meaning that this curve is the first, actually the unique, with all these characteristics

Secp256k1 was constructed in a special nonrandom way to ensure efficient computations. Here, follows the parameters defining the curve in DeFiS:

The finite field Fp is defined by the pseudo-Mersenne prime number
p = 2^256 − 2^32 − 2^9 − 2^8 − 2^7 − 2^6 − 2^4 − 1

```
// core/ecc.cpp
const uintBig Point::s_FieldOrder = {
// p = fffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f

0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, 0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFE,0xFF,0xFF,0xFC,0x2F
};
```

The defining equation E: y^2 = x^3 + ax + b is determined by:
a = 0, b = 7
Hence E: y^2 = x^3 + 7
The point G in compressed form is:
G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798
and in uncompressed form:
G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A68 5419 9C47D08F FB10D4B8

The order n of G:

```
// core/ecc.cpp
const uintBig Scalar::s_Order = {
 // n =
fffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd03
64141
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0xBA,0xAE
,0xDC,0xE6,0xAF,0x48,0xA0,0x3B,0xBF,0xD2,0x5
E,0x8C,0xD0,0x36,0x41,0x41};
```

The following 3rdparty/secp256k1-zkp primitives are used directly in the code:

- Curve point arithmetics secp256k1_gej
- Scalar arithmetics secp256k1_scalar
- SHA-256 Hash secp256k1_sha256_t
- Message authentication HMAC secp256k1_hmac_sha256_t
- Nonce generation secp256k1_nonce_function_rfc6979

Schnorr signatures

The signatures used in Bitcoin and Ethereum is called ECDSA which was as variant of Schnorr Signatures. One of the great advantages of Schnorr Signature over ECDSA is linearity, which means that multiple Schnorr signatures signed by different priv. keys for same

message can be verified by the sum of all their corresponding pub. keys.

Let Alice wants to sign the message M using private key k, and the public key C = k*G is publicly known.

Alice generates a random nonsense k1 and calculates C1 = k1*G.

Alice calculates the challenge by formula e = H(C1 | M).

Alice calculates k2 = k1 + e*k

Signature: (C1, k2)

Bob checks the signature:

Bob calculates the challenge by formula e = H(C1 | M).

Bob checks: C1 = k2*G - e*C

- This scheme is based on a noninteractive scheme where both sides can independently calculate the call e.
- The formula necessarily considers both the signed message and a part of the signature C1.
- The size of the signature can be compressed. For example, instead of C1 it can only contain its x coordinate or its hash function (i.e. H(C1)). At the same time, of course, the formula for calculating the e call must be changed accordingly, and use only what is in the signature.

Commitments

Suppose we have a v value that we want to encode. Strictly speaking, v can be anything, but it's not a randomly chosen value (random), but some parameter that an attacker can try to guess, and in practice it can be a relatively small number (compared to 256-bit).

So, if we encode it according to the standard scheme v*G, it can be easily broken.

To solve it, we expand our task. Until now, we worked with 1 generator G. Now we add an additional generator: the point H which is well known. The value is encoded through an expression: C = kG + vH.

- k is a private key (as before).
- v is the value we encode

It's called a commitment. It has the following properties:

- Hiding: no information about v can be obtained from this expression.
- Binding: If someone has generated a C, they cannot get the same C using other k,v. If you have to reveal its contents as a consequence, you will have to disclose these k,v.

Notes:

- It is important that no one knows the relationship between G and H, so H = x*G. Otherwise, the Binding is lost, as k,v can be changed while maintaining the same.
- For this reason, we can't just claim that the currents G,H are randomly chosen by us, may suspect that we took H = x*G, where x is known only to us.
- We must reveal the pattern in which the G,H points were selected. For example, you can take different lines and run them through a hash function to get points G,H.

So, Lets C = k*G + v*H, Alice wants to prove that she knows its contents without disclosing any details.

Alice generates random k1,v1.

Alice calculates C1 = k1*G + v1*H.

Bob sends challenge e

Alice calculates and sends k2 = k1 + e*k, v2 = v1 + e*v.

Bob checks: k2*G + v2*H = C1 + e*C.

- It's easy to see that the scheme with 2 generators and 2 parameters is also homomorphic, C(k1+k2,v1+v2) = C(k1,v1) + C(k2,v2).
- The transcript of the dock is a declaration of the dock for one parameter, and it can also be turned into a noninteractive scheme, and for a more compact form it is sufficient to use C1.x instead of C1.

Range proof and bulletproof

By means of Pedersen commitment it is possible to effectively codify values v. But it is important to remember the following nuances:

- You can "change" them without knowing the content. That is, knowing C(k,v) you can easily create C(k+dk,v+dv) = C(k,v) + C(dk,dv).
- The value v can be any value in the range [0 - p].

Next we will see v is interpreted as an amount, and during transactions are added/declined, respectively, added/declined declared amounts. Therefore, it is important that the value of v is limited and that overflow does not occur during arithmetic operations. In fact, values v close to p are equivalent to negative values, which should not be allowed.

There are schemes for proving that v is in a certain range, with no information disclosed other than this statement itself.

The Pedersen commitment in conjunction with the Range proof allows you to check the following:

* Proof that v is within a certain range of values.

* Proof that the creator of the range proof knows the disclosure, and that no one has changed it since.

MimbleWimble uses the Range Proof which proves that the value of v is in the range [1, 2^64]. This practically gives a large enough range of values for one value, while leaving a large stock of values that can be safely summed up.

The Bulletproofs technology is a Non-interactive Zero-knowledge (NIZK) proof protocol for general Arithmetic Circuits with very short proofs (Arguments of Knowledge Systems) and without requiring a trusted setup. They rely on the Discrete Logarithm (DL) assumption.

In Mimblewimble, the blockchain grows with the size of the UTXO set. Using Bulletproofs as a drop-in replacement for range proofs in confidential transactions, the size of the blockchain would only grow with the number of transactions that have unspent outputs. This is much smaller than the size of the UTXO set.

Oracle

Oracle is used in noninteractive cryptographic proof, it must create cryptographic challenges in a deterministic way, based on the visible transcript by the time.

Oracle uses hash in the straightforward way. The entire visible transcription is hashed. Once the challenge is needed - the hash value is finalized, the result is the challenge, and it's immediately re-fed to the Hash. So that the new challenge construction (if needed) is generated from the visible transcript, including the previous challenge.

If there are limits to the challenge (e.g. it must be a non-zero scalar or a valid x-coordinate of a curve point), - Finalize-Re-feed is called in the loop until a satisfactory challenge is produced (i.e. the accept/reject strategy is used).

Nonce generator

NG used in cryptographic proofs, but, unlike Oracle, the nonce generation involves secret data, and should not be possible to reconstruct by others.
Nonce generator is a combination of an Oracle, and the nonce function initialized by the secret data. That is, the Oracle accounts for all the visible transcript.

When a nonce is needed - first it's received from the Oracle, and then passed as an input to the nonce function (implemented in (**secp256k1**), which also uses the secret data. The final nonce generation function implemented in **secp256k1** actually a modified HMAC-SHA-256 scheme.

SBBS channels

The main goal of BBS is to allow wallets to communicate with each other in a secure and asynchronous manner. Using BBS wallets allows individuals to exchange messages, even if one of the individuals is offline. In general, BBS is a virtual board, where users can place messages, and each message is encrypted. For encryption, the public key of the recipient is used. This implies that the recipient's public key is his address in terms of this system. Every participant who is interested in messages from this board, observes and tries to decrypt new messages with his private key, and he manages to do so only if the message has been addressed to him. It consists of server and client sides. The server is implemented as a part of the node. The client is a wallet.

High-level protocols

Confidential transactions

Mimblewimble is a privacy-oriented, cryptocurrency technology. It differs from Bitcoin in some key areas:

- No addresses. The concept of Mimblewimble addresses does not exist.
- Completely private. Every transaction is confidential.
- Compact blockchain. Mimblewimble uses a different set of security guarantees to Bitcoin, which leads to a far more compact blockchain.

Confidential transactions use Pedersen Commitments to hide the value of a UTXO. In Mimblewimble, a transaction output or input is represented as a Pedersen Commitment rG + vH. G and H are random points on an elliptic curve and are public parameters of the blockchain. The value v is the UTXO value and r is the blinding factor and functions as the secret key for the UTXO. The value rG is the corresponding public key. Mimblewimble uses Pedersen commitments to obfuscate sensitive transaction information instead of showing plaintext transaction values. Pedersen commitments permit the use of basic arithmetic to validate transactions. By verifying that output commitments minus input commitments equal zero, we can confirm that no new money was created without knowing the actual input and output values. This works only if the values of the inputs sum to the value of the output and the blinding factor of the inputs sum to the blinding factor of the output.

TRANSACTION KERNEL The problem with confidential transactions as outlined above is that they require the input and output UTXO to use the same blinding factor, which is the recipient's secret key. If the sender learns the value of the recipient's blinding factor, she can steal the recipient's output UTXO. Mimblewimble overcomes this problem using zero-knowledge proofs. Consider a simple example of sending 5 coins. The sender has an unspent UTXO represented by the commitment X=45G+5H, where 5 is the value and 45 is her blinding factor (r), or secret key. The recipient chooses a random blinding factor 7 and creates an output UTXO represented by the commitment Y=7G+5H. A verifier that compares inputs to outputs will see the commitment of the excess: X-Y = (45G+5H) - (7G+5H) = 38G Mimblewimble calls the value 38 the excess or kernel, and the value X-Y = 38G the transaction kernel. In a valid transaction, the transaction kernel is always of the form X-Y = rG+0H, where r is some integer. This is true even if multiple inputs and outputs are used, If the sum of the input values is equal to the sum of the output values, the value multiplied by H will be zero. A valid transaction kernel is always in the form of a public key. The sender and receiver each know part of the corresponding

secret key. Mimblewimble has a protocol which lets them jointly compute a signature using their blinding factors to sign the transaction. The kernel represents a multisig key for transaction participants.

Coinjoin

One way to combat the public nature of transactions is CoinJoin. CoinJoin is a way to combine inputs into a single large transaction that makes it is difficult to distinguish which inputs are paying which outputs. CoinJoin has been implemented in JoinMarket, ShufflePuff, DarkWallet, SharedCoin, Wasabi, Samourai. The downside of wallet-based CoinJoin is that users have to opt-in to use the service. This diminishes its effectiveness because users either aren't aware of these services or don't care enough to go through the trouble of using them, resulting in a small set of CoinJoined transactions (a small "anonymity set"). This does not effectively hide originating addresses and destinations. Additionally, users must interact to create CoinJoin transactions since every input owner must sign the entire combined transaction to authenticate it. In MimbleWimble, users don't need to opt in CoinJoin is enabled by default. A block no longer has individual transactions. Rather, it looks like one large transaction. Figure 1 is a simplified version of an untouched set of transactions to be included in the next block. MimbleWimble joins the transactions together in a process similar to CoinJoin so that what is left is a single transaction that has combined a list of all inputs and a list of all outputs

Cut-trough

Cut-through removes outputs from the transaction pool, which have already been spent as new inputs, using the fact that every transaction in a block should sum to zero. This is shown below:

output−inputs=kernel−excess+(partof)kernel−offset

The kernel offset is used to hide which kernel belongs to which transaction and we only have a summed kernel offset stored in the header of each block.

We don't have to record these transactions inside the block, although we still have to record the kernel as the kernel proof transfer of ownership to make sure that the whole block sums to zero, as expressed in the following formula:

sum(ouputs)−sum(inputs)=sum(kernel−excess)+kernel−offset

An example of cut-through follows:

```
I1(x1)     +---> O1
           +---> O2

I2(x2,O2)  +---> O3

I3(O3)     +---> O4
           +---> O5
```

After cut-through:

```
I1(x1)     +---> O1
I2(x2)     +---> O4
           +---> O5
```

In the preceding examples, "I" represents new inputs, "X" represents inputs from previous blocks and "O" represents outputs.

This causes Mimblewimble blocks to be much smaller than normal bitcoin blocks, as the cut-through transactions are no longer listed under inputs and outputs. In practice, after this we can still see there was a transaction, because the kernel excess still remains, but the actual hidden values are not recorded.

Dandelion++

Dandelion is a transaction broadcasting mechanism that reduces the risk of eavesdroppers linking transactions to the source IP. Moreover, it allows XGM transactions to be aggregated (removing input-output pairs) before being broadcasted to the entire network giving an additional privacy perk.
Mechanism Dandelion transaction propagation proceeds in two phases: first the “stem” phase, and then “fluff” phase. During the stem phase, each node relays the transaction to a single peer. After a random number of hops along the stem, the transaction enters the fluff phase, which behaves just like ordinary flooding/diffusion. Even when an attacker can identify the location of the fluff phase, it is much more difficult to identify the source of the stem.
This mechanism also allows XGM transactions to be aggregated during the stem phase and then broadcasted to all the nodes on the network. This result in transaction aggregation and possibly cut-through (thus removing spent outputs) giving a significant privacy gain similar to a non-interactive coinjoin with cut-through.

Confidential Assetchain’s

Confidential Assetchain’s is much better feature then assets (tokens). It is a new cryptocurrency with own Blockchain that hold the characteristics of the parent chain (with the ability to change all parameters), but are also completely independent of the parent chain. Assets (aka tokens) are not cryptocurrencies, they haven't own blockchain and locked to the parent blockchain. Assets completely depends on the parent chain. By far the most popular platform to build tokens is the Ethereum network. However, there are others - Tron, Stellar, Omni, Waves, etc. This means that all tokens operate on the specific Blockchain they run on and their transactions are recorded in the Ledger that this specific Blockchain is. The fees for the transactions for the assets (tokens) on specific Blockchain are paid in the Coin that owns the Blockchain. This is why, when you have ERC-20 wallet and you have for example BAT in it and you need to send a transaction, you are required to have a small amount of ETH in the wallet in order to pay the "Gas" for the transaction.

The Confidential Assetchain’s are a runtime fork of XGM. It means that the source code of the executable binary remains same, it's just that the execution parameters are changed when you execute this binary and it creates its own independent blockchain. Users can set a new coin name (will be using in all functions by node, wallet, own db), set pre-mine in the first block, taxes, set difficulty algo params, mining algo, ALL params and rules. Assetchain’s can automatically receive XGM updates. So, Confidential Assetchain’s:

- Does not need XGM blockchain.
- Does not issue an asset (aka token) on XGM blockchain.
- Does not require $XGM coins to keep it running.
- Is absolute freedom to govern your own blockchain project.

Smart contracts

Scriptless scripts

Scriptless scripts are a way to encode smart contracts into digital signatures. This has applications way beyond Mimblewimble. The entire purpose is to do things without any scripts. Scriptless scripts are completely agnostic to which blockchain they are on, but they do require some sort of digital signature support. This was kind of hinted at during the panel earlier today. What we're doing here is removing these hash preimages, removing these various different script tricks that people use to get atomicity between chains and transactions, and moving those into signatures so that you get something smaller, more efficient, more scalable, more private, and also it's inherently interoperable.
Aaron Van Wirdum provides a great explainer, which we adapt here. He uses the example of a streamer who wants to listen to an artist’s song. The artist and streamer need to submit their combined Schnorr signature to the blockchain to validate the conditional transaction. The artist, who has the rights to the song

has a secret song key, 7000. The artist's half of the Schnorr signature is 8000. The artist can create an "adaptor signature" of 1000 by subtracting the secret song key (7000) from her piece of the Schnorr (8000). The artist then shares the adaptor signature with the streamer who uses cryptographic tricks to confirm that it equals the artist's piece minus the secret key. The streamer then shares her piece of the Schnorr signature with the artist. Let's say it's 5000. The artist submits the combined signature (8000+5000=13000) to the blockchain, automatically revealing her signature (8000) to the streamer. The streamer can now back into the secret song key (8000-1000=7000) to listen to the song. This all happens off-chain such that no one besides the artist and streamer ever discovers the individual values and steps. The only thing validators see is the combined Schnorr signature of 13000. Adaptor signatures are undetectable by the public. Nothing other than the "settlement transaction" is recorded on the blockchain.

Schnorr multi-signatures

One of the advantages of Schnorr's signature is that it can easily be generalized in the event that N participants want to collectively sign the message M, and the size of the signature does not depend on the number of participants. This is possible because the signature is essentially a scalar and a dot on a curve, both of which form an addition group. In other words, a multi-signature is essentially the sum of the signatures.

The only nuance is that all the participants in the signature and, consequently, the verifier use a single e call.

Let N participants, designated as P[i], want to sign the message M using the private keys k[i], and the public keys C[i] = k[i]*G are generally known.

* Each of the participants P[i] generates a random nonsense k1[i], and calculates C1[i] = k1*G.

* Participants sum up the received C1[i]. The result is C1.

* Each participant calculates the challenge by formula e = H(C1 | M).

* Each participant calculates k2[i] = k1[i] + e*k[i].

* Participants will summarize their k2[i]. The result is k2.

Signature: (C1, k2)

Vitalik checks the signature:

Vitalik calculates the challenge by formula
e = H(C1 | M).

Vitalik sums up all the public keys C[i].
The result is C.

Vitalik checks: C1 = k2*G - e*C.

- It's easy to see that this scheme is a summary of a single signatory case.
- As with a single signatory, the signature can be shortened by using C1.x or H(C1).
- Each of the signers sees what they are signing (message M), and part of the signature cannot be used for anything else (i.e. steal the signature).
- Signers may not know who they're signing a message with together, in a sense it gives them some kind of anonymity. If a member's signature is needed to confirm that he or she knows the line-up of signers, the formula for challenge e must be changed so that it explicitly uses all the member's public keys.

Adaptor signatures

Adaptor signatures is designed to communicate an extra piece of secret information between two parties

through multi-sig. Remember that Alice and Bob need to exchange the public side of the ephemeral keypair **Ra** and **Rb** before a joint aggregated multi-sig is created. If Bob also wants to sell a piece of secret information tb to Alice, the idea is that he can create an extra ephemeral keypair **(tb, Tb)** where **Tb=tbG** and communicate **Tb** to Alice alongside **Rb**. Right now, Bob is able to create a valid signature that includes **(tb, Tb)**:

```
bobSignature = (sb, R, Tb)
    where
        sb = kb + tb + ex
        e  = H(P||R+Tb||m)
        P  = Pa + Pb
        R  = Ra + Rb
```

tb is called an adaptor and it offsets the value kb a little bit. If we substract tb from sb, what we get is called an adaptor signature:

```
sb' = kb + ex
        = sb - tb
    where
        e = H(P||R+Tb||m)
        P = Pa + Pb
        R = Ra + Rb
```

If Bob sends **sb'** to Alice, Alice can verify three things:

```
-- 1) sb' is not a valid signature, since Tb is added to R
sb' = kb + ex
    where
        e = H(P||R+Tb||m)
        P = Pa + Pb
        R = Ra + Rb

-- 2) if e is replaced with e', then sb' is a valid signature
sb' = kb + e'x
    where
        e' = H(P||R||m)
        P = Pa + Pb
        R = Ra + Rb

-- 3) Tb's secret key tb is needed for a valid signature sb
sb = sb' + tb = kb + tb + ex
    where
        e = H(P||R+Tb||m)
        P = Pa + Pb
        R = Ra + Rb
```

Number 3) gives Alice the confidence that if she somehow learns **sb**, she will learn **Tb**'s secret key tb by substracting **sb'** from **sb**. She then feels comfortable to send her coin to a multi-sig address jointly owned by **Pa** and **Pb**, along with her signature (in reality, Bob might disappear, so it's important for Alice to have recourse to that, but that is not interesting for this discussion):

```
signatureAlice = (sa, R, Tb)
    where
        sa = ka + exa
        e  = H(P||R+Tb||m)
        P = Pa + Pb
        R  = Ra + Rb

-- created by Bob to take the coin from the multi-sig address
signatureJointMultiSig = (s, R, Tb)
    where
        s = sa + sb
            = sa + sb' + tb
        R = Ra + Rb
```

Using **signatureAlice**, Bob can create **signatureJointMultiSig** and use it to take the coin from the jointly owned multi-sig address that Alice just paid. As soon as **signatureJointMultiSig** hits the blockchain, secret tb is automatically revealed to Alice with the simple formula of **tb = s - sa - sb'** in a trustless manner.

Atomic cross-chain swaps

Alice and Bob can also use adaptor signatures to accomplish atomic swap by executing essentially the same protocol in parallel on two chains.

Let's say **ChainA** and **ChainB** use the same elliptic curve (e.g. secp256k1). Alice and Bob agree to swap Alice's coin on **ChainA** with Bob's coin on **ChainB** in a trustless and atomic way. Assuming that Alice and Bob's keypair is **(**xa_A**, Pa_A)** and **(**xb_A**, Pb_A)** on **ChainA** and **(**xa_B**, Pa_B)** and **(**xb_B**, Pb_B)** on **ChainB** respectively. First of all, Alice's coin will be paid to a multi-sig address controlled by **Pa_A** and **Pb_A** on **ChainA** while Bob's coin will be paid to a multi-sig address controlled by **Pa_B** and **Pb_B** on **ChainB**. The mechanism for getting refund when counterparty disappears is intentionally ignored here for brevity.

Next, Alice and Bob exchange the ephemeral keypairs on both chains, to follow the same convention, **(**ka_A**, Ra_A)**, **(**kb_A**, Rb_A)** on **ChainA** and **(**ka_B**, Ra_B)**, **(**kb_B**, Rb_B)** on **ChainB**. On top of that, Bob generates an extra ephemeral keypair **(**tb**, Tb)** and share it with Alice on both chains. Bob's signature could eventually look something like this:

```
bobSignatureChainA = (sb_A, R_A, Tb)
  where
    sb_A = kb_A + tb + e_Axb_A
    e_A  = H(P_A||R_A+Tb||m)
    P_A  = Pa_A + Pb_A
    R_A  = Ra_A + Rb_A

bobSignatureChainB = (sb_B, R_B, Tb)
  where
    sb_B = kb_B + tb + e_Bxb_B
    e_B  = H(P_B||R_B+Tb||m)
    P_B  = Pa_B + Pb_B
    R_B  = Ra_B + Rb_B
```

However, just like what Bob did before, he sends Alice his adaptor signatures **sb_A' =** kb_A **+ e_A**xb_A and **sb_B' =** kb_B **+ e_B**xb_B for her to verify. After Alice feels confident that **Tb**'s secret key tb is the one needed to get **sb_A** and **sb_B** from **sb_A'** and **sb_B'**, she sends her part of the multi-sig signature on ChainA to Bob:

```
aliceSignatureChainA = (sa_A, R_A, Tb)
  where
    sa_A = ka_A + e_Axa_A
    e_A  = H(P_A||R_A+Tb||m)
    P_A  = Pa_A + Pb_A
    R_A  = Ra_A + Rb_A
```

Bob can produce an aggregated signature **signatureJointMultiSigChainA** by combining **aliceSignatureChainA** and **bobSignatureChainA** to take Alice's coin on ChainA, as shown below:

```
signatureJointMultiSigChainA = (s_A, R_A, Tb)
  where
    s_A = sa_A + sb_A
        = sa_A + sb_A' + tb
    R_A = Ra_A + Rb_A
```

At the same time, knowing **sb_A'**, Alice can calculate the secret value tb by substracting **sa_A** and **sb_A'** from **s_A**. Since Alice also knows **sb_B'**, she can also use tb to calculate a valid joint signature **signatureJointMultiSigChainB** on ChainB to take Bob's coin, accomplishing the process of atomic swaps.

```
aliceSignatureChainB = (sa_B, R_B, Tb)
  where
    sa_B = ka_B + e_Bxa_B
    e_B  = H(P_B||R_B+Tb||m)
    P_B  = Pa_B + Pb_B
    R_B  = Ra_B + Rb_B

signatureJointMultiSigChainB = (s_B, R_B, Tb)
  where
    s_B = sa_B + sb_B
        = sa_B + sb_B' + tb
    R_B = Ra_B + Rb_B
```

Again, from the perspective of the blockchain and the rest of the world, only simple signatures are involved in those transactions, greatly improves privay, fungibility and efficiency,

Atomic multi-path payment

Atomic multi-path payments solve the problem is that you need a path on the network between multiple nodes on the graph. The problem is that if Alice wants to send 80 XGM (or any PAXXX) she has to these numbers are the capacities in the direction towards Bob. There's a capacity in both directions. If Alice wants to pay Bob she wants to send 80 XGM but she can't because each path on its own doesn't have enough capacity. And Bob at a time can only receive up to 100 XGM because he has that inbound liquidity, but he's unable to because of the single path constraint.

Atomic multi-path payments allow a single logical payment to be sharded across multiple paths across the network. This is done at the sender side. Multiple subtransactions can be sent over the network and take their own paths and then the necessary constraint is that they all settle together. If one of the payments fails then we want them to all fail. This is where the atomicity comes in.

This enables better usage of in-bound and out-bound liquidity. You can send payments over multiple routes, and this allows you to split up and use the network better. This is a more intuitive user experience because like a Bitcoin wallet you expect to be able to send most of the money in your wallet. Without Atomic multi-path payments, this is difficult because you can only send a max amount based on current channel liquidity or something, which doesn't really make sense to a user.

Only the sender and receiver need to be aware of this technology. For this reason, it can be adopted into the current lightning network scheme and for other parties they look like single normal payments.

Discreet Log contracts

Smart contracts are an often touted feature of cryptographic currency systems such as Bitcoin, but they have yet to see widespread financial use. Two of the biggest hurdles to their implementation and adoption have been scalability of the smart contracts, and the difficulty in getting data external to the currency system into the smart contract. Privacy of the contract has been another issue to date. Discreet Log Contracts are a system which addresses the scalability and privacy concerns and seeks to minimize the trust required in the oracle which provides external data. The contracts are discreet in that external observers cannot detect the presence of the contract in the transaction log. (https://adiabat.github.io/dlc.pdf)

Discreet Log Contracts (DLC) in our way combined with Adaptor signatures (AS). In DLC, the oracle will reveal one of multiple possible s values as part of signing the outcome of an event. This s is essentially a private key for which the public key S can be calculated ahead of time (because R is committed to in advance).

In AS, instead of just R (essentially a public key), you add a second public key P of which the payer wishes to obtain the private key p from the payee. Only by revealing p can the payee make the signature valid, and thus receive the payment.

If we use S in place of P, we have essentially combined DLC and AS.

Example:

Alice and Bob (A and B) want to bet 100 XGM on whether it will rain tomorrow.

Olivia will publish "yes" or "no" under her key O and commitment R.

This means there are two possible values for S:

S1 = R + hash(R, "yes")*O

S2 = R + hash(R, "no")*O

Alice and Bob create a payment channel under key A + B = C with 100 XGM each.

They propose two possible channel updates: 200 XGM for Alice if it rains, or

200 XGM for Bob if it doesn't.

The channel update (simplified to single key C) where Alice wins is signed

with:

R1 = r*G + S1

s' = r + hash(R1, transaction)*c

Note that we wrote s' because s is not complete. We added S1 to R, so we

need to add s1 to s' in order to get s.

And similarly for Bob:

R2 = r*G + S2

s' = r + hash(R2, transaction)*c

Let's say Bob was right and Olivia signs "no", thereby revealing s2. This

now completes the signature: s = s' + s2.

s*G == R2 + hash(R2, transaction)*C

Multi Oracle

Let's say that you want to use oracle 1 and oracle 2, that's easy, you just add up their points. It has to be both signing the same thing. And there's no size increase. You can do m-of-n. We need two of these three oracles to sign it, but it starts to blow up the size of the state between the two parties, it gets really big really fast. The other tricky part here is that you have to sign the same exact thing. If one oracle signs 52 and the other one signs 53, and they didn't intend to sign different values, then you can't close your transaction because none of your things will add up. There can be timeout transaction where if the oracle goes down, then after a week or two, Bob and Alice should each be able to get half of their money back. If the oracle dies then it's a wash trade and they should revert.

DeFiS Mechanics

The required set of system modules will include:

- Creating confidential crypto asset assets
- Decentralized tied assets (pegged coins enable work with crypto assets of various blockchains within the same system)
- Decentralized oracle pricing (used to collect data from other blockchain networks and from centralized resources)
- Decentralized exchanges (allows atomic swaps, exchanges crypto assets within and between various blockchains)

DeFiS Assetchain Assets (DAA)

DeFiS is built on its own blockchain platform and uses its XGM cryptocurrency as the main service tool within the system. There are 2 types of DAA Assets in DeFiS (DeFiS Assetchain Assets):

· Custom Assetchain Asset (CAA)

· Tethered Pegged Assetchain Asset (PAA)

Custom Assetchain Asset (CAA)

CAA is a user asset chain asset that can be created by any user to present any project or a set of smart contracts (for the purpose of asset co-investment or the creation of stablecoins). The set of functions and parameters for such assets is constantly expanding (both paid and free functions are planned). Key CAA creation options:

- Ticker / cryptocurrency name
- Block reward
- Distribution scheme (total issue of coins / pre-mine / type of issue)
- Ability to create tokens inside your blockchain
- Inclusion of add. scalability
- Mixed mining
- POW algorithm
- Enabling protocols for finance selectively
- Technical parameters (block size, network confirmation time, initial difficulty, etc.)

Pegged Assetchain Asset

An important task for DEFIS is the ability to work with various crypto assets directly. PAA-linked assets to major external cryptocurrencies are created and maintained decentralized. They use technology of confidential assetchain, atomic swaps, Hash Time Locked Contract and Multisig for reliable use and communication with external crypto assets (by blocking / releasing and creating tied (pegged) assets).

- PABTC linked to BTC
- PAETH linked to ETH.
- PAXRP linked to XRP
- PAUSDT linked to USDT
- PABCH linked to BCH, etc.

Personalized Assetchain Debt Contract (PADC)

Personalized Debt Contract (PADC) is designed to allow the PADC owner to take a secured loan secured by

PADC. Each PADC is unique to each address. Any user can open PADC for free. The ownership of the contract may be transmitted.

Once the PADC is open, the owner can send XGM to him to finance the collateral. After that, PADC allows the owner to take a loan, having minted PAA in the amount of up to a certain coefficient from the collateral. The minimum collateral ratio starts at 150%.

In other words, a security deposit of $ 1,500

USA (in XGM), allows the PADC owner to take a loan for a maximum of $ 1,000. Intermediate PAAs are subject to a floating borrowing rate. PADC doesn't have expiration date. The owner can take a loan as much as he wants, provided that the collateral ratio is always above 150%.

Collateral ratio = collateral / (Credit + accrued interest).

The Personalized Debt Contract (PADC) is designed to allow the PADC owner to take a secured loan against the security secured by PADC. Each PADC is unique to each address. Any user can open PADC for free. The ownership of the contract may be transferred.

· Once the PADC is open, the owner can send XGM to him to finance the collateral. After that, PADC allows the owner to take out a loan, having minted PAA in the amount of up to a certain coefficient from the collateral. The minimum collateral ratio starts at 150%. In other words, a security deposit of $ 1,500 (in XGM) allows the PADC owner to take a loan of a maximum

of $ 1,000. Intermediate PAAs are subject to a floating rate of borrowing. PADC has no expiration date. The owner can take a loan as much as he wants, provided that the collateral ratio will always be above 150%.

· Collateral ratio = collateral / (Credit + accrued interest).

Closing a PADC entitles its owner to receive back all 100% of the security deposit. To close the PADC, the owner must repay the loan in full, plus the accrued interest in his sub-currency PAA (for example, PABTC).

The main advantage of DeFiS in this scheme is the ability to pledge any asset thanks to the XACX cross-platform exchange mechanisms and the PAPD repository of linked assets in DeFiS.

The role of the PAPD is to maintain the price guarantee of the PAA to its actual asset, for example, PABTC to BTC, PAETH to ETH, etc.

PAPD depositories are not personalized and act as depositories that collectively hold all of the collateral from PADC.

PAPD sets the base purchase and sale price of PAA on DEX at the spot rate aggregated from oracle pricing contracts, provided that the PAA has enough collateral / PAA in its depository to cover it.

Pegged Assetchain Assets repository (PAPD)

PAPD starts without PAA, but with XGM as collateral for PADC. As long as there is enough XGM in the PAPD, and as long as the PAPD has less than the total number of issued XGMs, the PAPD will place the following purchase orders on DEX: Buy PABTC for the price of 100,000 XGM (i.e. $ 10k). Buy PAETH for the price of 2000 XGM (i.e. $ 200).

If PABTC and / or PAETH are sold by PAPD, then PAPD will place the following orders, provided that it has PAA at its disposal:

Sell PABTC for 100,000 XGM (i.e. $ 10k) Sell PAETH for 2000 XGM (i.e. $ 200). Regardless of whether he buys or sells PAPD, transactions are always for him without commissions on DEX because a non-PAPD party pays a commission.

DeFiS Decentralized Exchange

The internal DEX of the DeFiS system ensures decentralized trading of all assets in the system and XGM itself, which means that all crypto assets: XGM and DAA (PAA and CAA) will be traded on DEX. Initially, DEX will be launched with XGM as the base trading pair, providing markets such as PABTC / XGM, PAETH / XGM, PAUSDT / XGM, etc. With the increase in volume, other basic trading pairs, such as PAETH / PABTC, etc. can be introduced.

PAPD also trades in DEX automatically, setting a base price for the PAA.

XACX Cross platform Assetchain exchange

A user who owns PABTC may be interested in having the actual BTC instead of PABTC. DEFIS Cross-chain Exchange (XACX) is designed for this. XACX allows the exchange of PAA with its sub-asset, for example, PABTC for BTC, PAETH for ETH, PAXRP for XRP.

Pricing Oracle

A pricing contract is a smart contract system that allows several trusted and designated parties to provide periodic PAA and XGM price lists.

DeFis Use-cases

Using a long position for investing (profit on growth)

Bob has a 100k XGM. He believes in the growth of XGM and he wants to strengthen his position in this asset for the long term.

- Bob opens PADC in DeFiS and takes a loan in PAUSDT
- Bob buys more XGM for PAUSDT
- Thus, Bob gains a composite long position on XGM without additional investments

Using a short position for investment (profit on the fall)

- Bob wants to “shorten” the XXX asset. Bob has XGM
- Bob opens PADC at DeFiS and takes out a PAXXX loan with XGM as collateral
- Now Bob can either sell PAXXX for XGM or PAUSDT on DeFiS DEX, or convert PAXXX through XACX cross-platform DeFiS to XXX exchanges to sell XXX on any other exchange.
- As soon as Bob wants to close a short position (XXX has fallen in price), Bob buys back XXX (or PAXXX) to the market at a lower price and closes its PADC

Getting a loan

- Bob has XGM, but he needs a short-term cash infusion in another XXX asset. Bob does not want to sell XGM for her and does not want to spend money on the purchase of this asset
- Bob takes a loan through PADC to PAXXX and converts it to XXX.
- As soon as he wants to repay the loan, Bob just buys XXX / PAXXX and closes your PADC

Loans

- Bob has a BTC that he does not need in the short term. Bob wants to get some interest (cash flow) by issuing BTC on credit.
- Bob transfers BTC to the XACX cross-platform DEFIS exchange, indicating the amount of BTC, the desired premium (interest rate) and the period (period during which he does not need BTC).
- As soon as the counterparty uses Bob's offer, Bob makes an instant profit in XGM.
- At the end of the expiration period, Bob will receive his BTC back, or he will be able to receive XGM with an additional guarantee, thus earning more than his original amount in BTC.

XGM Tokenomics

XGM cryptocurrency is an integral part of the DeFiS ecosystem and is used to carry out any operations in a financial system.

Defis XGM mainnet is public since the first block and was launched at June 10, 2020. XGM is POW coin, it means network supporting by miners in decentralized way.

XGM is launched as Grimm hard fork. There 2 networks running after HF:

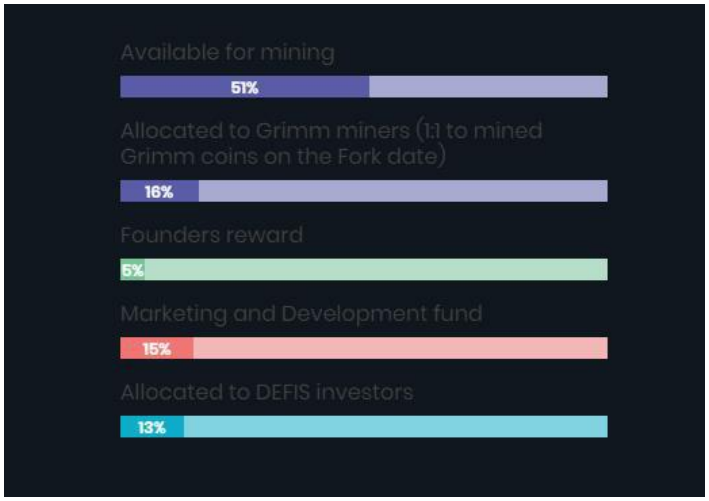
- Previous network Grimm $GRIMM
- New network Defis $XGM

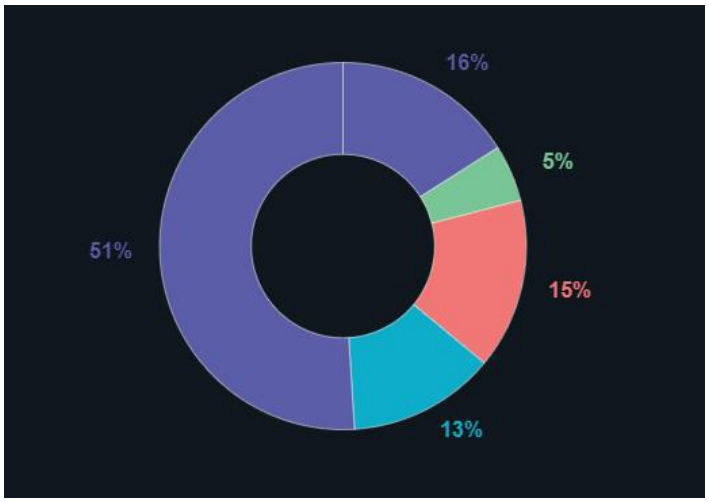
The XGM total supply is 262,800,000 XGM coins, or 26 279 999 976 873 600 Centums (1 XGM = 100 000 000 Centums)

Emission - Deflationary:

- At first 4 years XGM miner reward = 20 XGM per block
- In 5th year reward will be reduced to 10 XGM, and then will halving every 4 years

Coins allocation from network start:





- Grimm Holders was airdropped 1:1 GRIMM:XGM (16% from max supply) after launch Defis XGM
- Founders reward 5%
- Marketing and development fund 15%
- Allocated to private investors 13%
- Available for mining 51%

XGM is used in DeFiS:

- To pay for all transactions and smart contracts
- To pay rewards to miners for network support
- As payment of commissions for decentralized p2p exchanges
- To pay for DEX commissions
- To make XACX payments
- For interest payments on PADC
- As a collateral for borrowing other crypto assets
- To pay for additional CAA services (scaling, merge mining, cloud hosting nodes, etc.)

XGM Specifications

Protocol MimbleWimble

Language C++

Consensus PoW

PoW Algorithm spec. -GrimmPOW based on Equihash 150_5

Mining XGM wallet with built-in GPU and CPU mining and mining pools

External miners Gminer for Nvidia & AMD / MiniZ for Nvidia

Emission Deflationary

Block Reward 20 XGM. Rewards halving every 4 years.

Max Supply 262.8M

Governance Corporate

Blocktime 30 sec

Block size 4 Mb

Initial TPS 136 tx per sec

Smallest unit CENTUM (0.00000001 XGM)